ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

И. В. Черпаков

ОСНОВЫ ПРОГРАММИРОВАНИЯ

УЧЕБНИК И ПРАКТИКУМ
ДЛЯ ПРИКЛАДНОГО БАКАЛАВРИАТА

Рекомендовано Учебно–методическим отделом высшего образования в качестве учебника для студентов высших учебных заведений, обучающихся по экономическим направлениям

Книга доступна в электронной библиотечной системе biblio-online.ru

Москва ▪ Юрайт ▪ 2018

УДК 004.43(075.8)
ББК 32.973.26-018я73
Ч49

***Автор*:**

Черпаков Игорь Владимирович — кандидат физико-математических наук, доцент кафедры информатики, математики и общегуманитерных наук Липецкого филиала Финансового университета при Правительстве Российской Федерации.

***Рецензенты*:**

Кудинов Ю. И. — доктор технических наук, профессор, заведующий кафедрой информатики Липецкого государственного технического университета;

Петренко С. В. — доктор технических наук, доцент кафедры информатики и математических методов моделирования в экономике Липецкого государственного педагогического университета.

Черпаков, И. В.

Ч49 Основы программирования : учебник и практикум для прикладного бакалавриата / И. В. Черпаков. — М. : ИздательствоЮрайт, 2018. — 219 с. — Серия : Бакалавр. Прикладной курс.

ISBN 978-5-9916-9983-9

Задача этой книги — изложить элементы теории алгоритмов и основы программирования в рамках относительно несложной для изучения системы PascalABC.NET.

Книга содержит значительное количество справочного материала, большое число примеров, в том числе и экономической направленности. Рассмотрены основные структуры данных и технологии структурного программирования. В конце глав приведены тестовые задания и нетривиальные задачи для самостоятельного решения.

Отдельное место в книге занимает описание технологии объектно-ориентированного программирования с использованием .NET: рассмотрены основные понятия объектно-ориентированного подхода, работа с динамическими массивами, обработка исключений.

Соответствует актуальным требованиям Федерального государственного образовательного стандарта высшего образования..

Для студентов высших учебных заведений, обучающихся по экономическим направлениям и специальностям.

УДК 004.43(075.8)
ББК 32.973.26-018я73

ISBN 978-5-9916-9983-9

Оглавление

Предисловие

Современные информационные системы, составляющие основу управления экономическими объектами, позволяют накапливать и хранить достаточно большие объемы информации. Владение базовыми средствами и знание основных алгоритмов обработки данных, применяемых в подобных системах, является одним из требований, которым должны удовлетворять выпускники высших учебных заведений по направлению «бакалавр бизнес-информатики».

Учебник «Основы программирования» предназначен для студентов, обучающихся по направлению подготовки «бакалавр бизнес-информатики», при изучении дисциплин, связанных с элементами алгоритмизации, программирования, объектно-ориентированного анализа и проектирования и других смежных дисциплин. Также он может быть использован студентами других направлений подготовки в качестве дополнительного материала при самостоятельном изучении основ теории алгоритмов и программирования на языке PascalABC.NET.

Целью настоящей книги является изложение базовых положений теории алгоритмов и основ программирования в рамках решения экономических задач. Необходимость ее написания обусловлена фактом недостаточного количества изданий, отражающих связь теоретических основ программирования с практическими ситуациями, возникающими в экономической сфере.

Учебник состоит из 11 глав, каждая из которых содержит теоретический и практический материал, а также примеры, поясняющие новые понятия и особенности программного кода, связанного с рассматриваемыми вопросами. В теоретический материал включается описание синтаксиса для сред программирования PascalABC.NET и Turbo Pascal с указанием различий для каждой среды. Практический материал иллюстрирует возможности применения рассматриваемых понятий, в том числе и в задачах экономической направленности. Программный код алгоритмов задач снабжен подробными комментариями. К ряду заданий, связанных с обработкой массивов, приводятся блок-схемы алгоритмов решения.

В конце каждой главы (кроме гл. 10) приведены контрольные вопросы, тесты и задания для самостоятельного решения, которые

помогут оценить знания, полученные при чтении главы. Ответы на тесты приводятся в конце учебника. Глава 10 является вспомогательной: в ней рассматриваются только те понятия, которые позволяют понять материал следующей главы, поэтому она содержит только контрольные вопросы.

В результате изучения материала студент должен:

знать

- основные понятия теории алгоритмов, базовые алгоритмические конструкции;
- теоретические основы оценки алгоритмической сложности алгоритмов, выполненной на основе анализа программного кода;
- методологические принципы построения и реализации алгоритмов;
- основные языковые конструкции и структуры данных языка Pascal;
- основные подходы к организации сложных структурированных данных и способы их реализации;
- основные принципы структурного и объектно-ориентированного подхода в программировании;

уметь

- определять сложность алгоритмов на основе анализа программного кода;
- разрабатывать алгоритмы обработки данных на основе базовых алгоритмов, применяемых к данным различного типа;
- решать задачи, связанные с обработкой экономической информации, самостоятельно определять структуры данных, необходимых для решения;
- использовать технологии .NET при написании программ в среде PascalABC.NET;
- применять объектно-ориентированные технологии программирования при разработке приложений;

владеть

- технологиями разработки и оценки алгоритмов;
- навыками практического программирования в среде PascalABC.NET;
- принципами декомпозиции прикладных задач и реализацией отдельных элементов в виде подпрограмм, модулей, классов;
- базовыми технологиями объектно-ориентированного программирования для решения экономических задач;
- навыками разработки приложений с использованием технологий .NET.

Автор будет благодарен вашим отзывам о данном учебнике и сообщениям о найденных неточностях. Ваши предложения и замечания можно присылать на адрес электронной почты: cherpackov@gmail.com.

Глава 1
АЛГОРИТМ И АЛГОРИТМИЧЕСКИЕ СТРУКТУРЫ

1.1. Интуитивное определение алгоритма

Любая человеческая деятельность, в частности решение задач, связанных с реализацией информационных процессов, представляет собой последовательность действий, которую возможно описать с использованием языковых, графических и других средств. Попытка формально (по определенным, строго зафиксированным правилам) описать выполняемые действия приводит к понятию алгоритма.

В общем случае, под **алгоритмом** понимается точно сформулированная конечная последовательность действий (операций), выполнение которой приводит к некоторому результату. Такое определение является интуитивным, т.е. строго не определенным (неформальным).

Существуют и другие интуитивные определения, например — сформулированные А. Н. Колмогоровым и А. А. Марковым.

Алгоритм (по А. Н. Колмогорову) — это всякая система вычислений, выполняемая по строго определенным правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи.

Алгоритм (по А. А. Маркову) — это точное предписание, определяющее вычислительный процесс, идущий от варьируемых исходных данных к искомому результату.

Интуитивное определение позволяет представить процесс решения задачи как преобразование некоторого набора исходных данных (*входа*) в некоторый результатный набор данных (*выход*), осуществляемое *исполнителем алгоритма* в соответствии с некоторыми инструкциями (рис. 1.1).

Вход и выход характеризуются данными различной структуры. Например, входом алгоритма нахождения определителя матрицы является квадратная матрица порядка n (точнее, ее числовые эле-

Рис. 1.1. **Процесс решения задачи**

менты), а выходом — число. Исполнителем алгоритма может выступать любой субъект, который способен воспринять и исполнить инструкции алгоритма, например человек, группа людей, компьютер, абстрактная машина.

Различные подходы к понятию алгоритма, в том числе указанные выше, определяют ряд требований (также называемых *свойствами*), которым он должен соответствовать:

• правильность — при любых допустимых входных данных результат выполнения алгоритма должен удовлетворять требованиям задачи (результат должен быть правильным). Проверка этого свойства производится, например, серией тестовых данных с известными входными значениями и правильными выходными;

• дискретность — алгоритм должен представлять собой последовательность отдельных операций (предписаний, действий, инструкций);

• конечность — алгоритм должен записываться в виде конечного числа операций;

• элементарность операций — каждое действие не должно допускать неоднозначного толкования;

• определенность — после выполнения каждой операции однозначно известно, какое действие будет выполняться следующим;

• результативность — после завершения выполнения алгоритма известен результат, который может быть интерпретирован и записан в терминах решения задачи. Результатом считается также и выявление того факта, что решение задачи невозможно;

• однозначность — применение алгоритма к одним и тем же входным данным дает один и тот же результат.

Кроме указанных свойств, часто к алгоритму предъявляются требования, важные с практической точки зрения:

• универсальность (массовость) — алгоритм должен решать не конкретную задачу, а целый класс схожих задач. Другими словами, алгоритм должен быть единым для различных данных из допустимого множества входных данных. Это требование не является обязательным, оно отражает, скорее, качество самого алгоритма;

• разумное время исполнения. Если время исполнения алгоритма велико, то его практическая ценность мала. Например, многие алгоритмы, использующие полный перебор вариантов, даже

при современной скорости обработки данных должны выполняться сотни миллиардов лет, что сводит на нет их практическое значение;

• разумная ресурсоемкость. Технические характеристики (например, объем оперативной памяти, тактовая частота процессора) компьютерной техники должны быть достаточны для выполняемого алгоритма.

Основную сложность при разработке алгоритмов составляет выполнение двух последних требований. От того, насколько эффективным по времени исполнения и требуемым ресурсам будет алгоритм, зависят время получения результата и общая стоимость решения той или иной задачи.

1.2. Формы представления алгоритмов

Представление алгоритмов может быть выполнено в различных формах. Основными из них являются формульно-словесная, графическая, представление на языке программирования и с использованием псевдокода.

Формульно-словесная форма. Алгоритм представляется на естественном языке, возможно — с использованием математических символов, выражений и формул. Такая форма представления удобна для формулирования несложных или обобщенных алгоритмов. Используемые языковые конструкции строго не формализуемы, поэтому отдельные шаги алгоритма могут допускать неоднозначность толкования. Для сложных алгоритмов данная форма становится громоздкой и не наглядной.

Пример 1.1

Записать в формульно-словесной форме алгоритм решения уравнения $ax + b = 0$ с произвольными целочисленными коэффициентами.

Решение

Алгоритм решения уравнения приведен ниже.

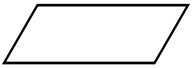
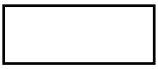
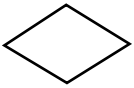
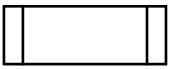
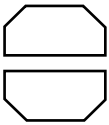
1. Задать начальные целые значения коэффициентов a, b.
2. Если $a \neq 0$, то перейти к шагу 3, иначе перейти к шагу 4.
3. Решением будет значение $x = -b/a$. Перейти к шагу 5.
4. Если $b = 0$, то решением будет любое число, иначе решений нет.
5. Закончить алгоритм. ■

Графическая форма. Алгоритм представляется в виде взаимосвязанных графических элементов. Связи, изображаемые также графически, отражают логику программы и последовательность выполнения отдельных шагов алгоритма. В качестве примера можно привести форму записи в виде блок-схемы.

Блок-схема представляет собой набор графических элементов (блоков), связанных соединительными линиями. В настоящее время в Российской Федерации действует ГОСТ 19.701—90 «Схемы алгоритмов программ, данных и систем», согласно которому следует оформлять блок-схемы. Основные блоки, указанные в этом стандарте, приведены в табл. 1.1.

Таблица 1.1

Основные блоки, используемые в блок-схемах

Графический объект	Описание
	Начало и конец алгоритма
	Начало и конец части алгоритма
	Соединительная линия. При необходимости или для повышения удобочитаемости могут быть добавлены стрелки-указатели
	Ввод или вывод данных для произвольного носителя данных
	Выполнение действий, вычислений; обработка данных любого вида
	Логический блок. Вход в блок обычно обозначается линией, направленной к верхней вершине
	Предопределенные действия, подпрограмма, описанная в другом месте программы или модуле
	Начало и конец циклических конструкций
	Комментарий

Пример 1.2

Записать в виде блок-схемы алгоритм решения уравнения $ax + b = 0$ с произвольными целочисленными коэффициентами.

Решение

Блок-схема приведена на рис. 1.2.

Для выполнения вычислений в примере использован синтаксис языка Pascal. ■

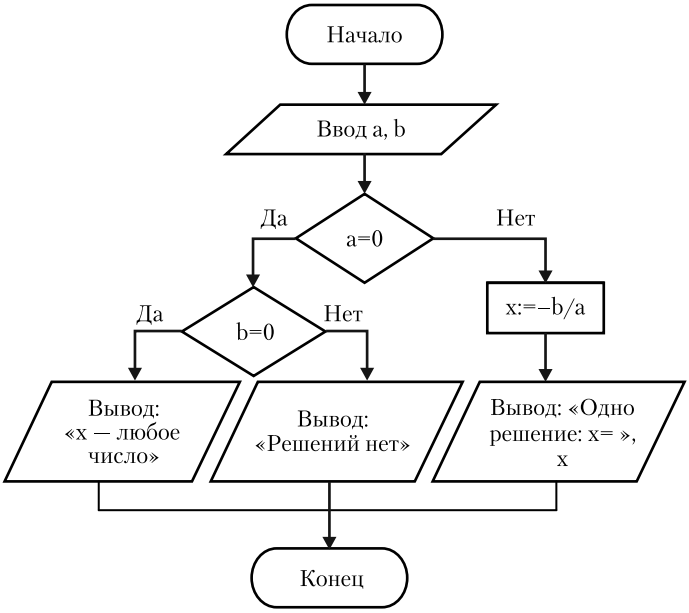


Рис. 1.2. **Пример блок-схемы алгоритма**

Алгоритм, записанный в виде блок-схемы, более нагляден, чем в формульно-словесной форме. В то же время графическая запись занимает много места, и в случае сложного алгоритма наглядность сильно снижается из-за большого количества соединительных линий.

На практике при использовании блок-схем применяют последовательную детализацию алгоритма: сначала составляется обобщенная (укрупненная) блок-схема, затем каждый из блоков представляется в виде отдельной блок-схемы и т.д.

Представление на языке программирования. Если первые две формы представления ориентированы, прежде всего, на пользователя, то данный способ — на машинную реализацию алгоритма.

Для записи последовательности действий используются конструкции некоторого искусственного языка. Алгоритм, записанный на этом языке, называется **программой**, а сам язык — **языком программирования**. Программа может непосредственно переводиться (транслироваться) в исполняемые процессором команды.

В настоящее время используется довольно большое количество языков программирования, каждый из которых имеет свою специфику. В основном они отличаются **синтаксисом** (правилами образования языковых конструкций) и **семантикой** (смысловым значением языковых конструкций). К наиболее популярным языкам можно отнести C (C++, C#, C), Java, PHP, Visual Basic, Pascal.

В листинге 1.1 приведена запись алгоритма, рассмотренного выше, на языке программирования Pascal.

Листинг 1.1

```
Program uravnenie;
Var
  a,b,x:real;
begin
write('Введите коэффициент a: '); Readln(a);
write('Введите коэффициент b: '); Readln(b);
if a=0 then
  if b=0 then writeln('x - любое число')
  else writeln('Решений нет')
else
  begin
    x:=-b/a;
    writeln('Одно решение: x=', x:5:2);
  end;
end.
```

Представление в виде псевдокода. Данная форма является промежуточной между записью на языке программирования, формульно-словесной формой и блок-схемой.

Псевдокод — это способ описания алгоритма, имеющий следующие особенности:

1) в его основе лежит некоторый язык программирования;

2) синтаксические конструкции на псевдокоде практически повторяют конструкции языка программирования, но несущественные подробности синтаксиса и семантики опускаются;

3) отдельные операции и их группы могут именоваться произвольно, иногда с помощью естественного языка.

Основная цель применения псевдокода — показать сущность алгоритма, чтобы, с одной стороны, он был понятен человеку, не знакомому со специфическим синтаксисом языка программирования, а с другой — минимизировать время записи алгоритма на этом (или другом) языке.

Выбор основы псевдокода связан с так называемой *выразительностью языка программирования* — степенью близости его синтаксических конструкций к естественному языку. Чем выше уровень языка программирования, тем обычно он более выразителен. Например, заголовок цикла на языке C:

```
for (i=1; i<=10; i++)
```

менее выразителен, чем на языке Pascal:

```
for i:=1 to 10 do…
```

Во втором случае запись очень похожа на естественный язык: «Для переменной i, которая изменяется от значения 1 до значения 10, выполнять...».

Записать алгоритм рассмотренной выше задачи с помощью псевдокода можно, например, следующим образом:

```
алг уравнение
вещ a,b,x
нач
  ввод a,b
  если a=0
    то
      если b=0
        то вывод «x — любое число»
        иначе вывод «решений нет»
      все
    иначе
      x:=–b/a
    вывод x
  все
кон
```

Каждая из приведенных форм представления алгоритмов имеет свои достоинства и недостатки. В любом случае выбор той или иной формы определяется сложностью решаемой задачи, выбором средств, с помощью которых будет реализовываться алгоритм, и удобством восприятия записанного алгоритма.

1.3. Базовые алгоритмические структуры

Теоретические исследования 1970-х гг. показали, что произвольный алгоритм может быть представлен как совокупность трех типов алгоритмических конструкций: следования, развилки, цикла. Они являются своего рода алгоритмическими единицами, в виде комбинации которых может быть составлен алгоритм. Данные конструкции получили название **базовых алгоритмических структур** (БАС). Каждая БАС имеет ровно один вход и ровно один выход.

Следование. Данная конструкция состоит из последовательно выполняемых действий (инструкций), в качестве которых могут выступать и другие БАС. На рис. 1.3, *а* приведена общая форма записи следования в виде блок-схемы. На рис. 1.3, *б* — конкретный пример, в котором в качестве действий указаны подпрограммы вычислений наименьшего общего кратного и наибольшего общего делителя.

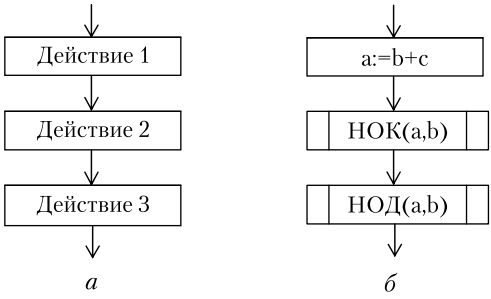


Рис. 1.3. **Следование**

Развилка. Другие названия этой БАС — ветвление, условие. Данная структура предусматривает проверку некоторого условия и выполнение тех или иных действий в зависимости от его истинности или ложности. Каждый из путей ведет к общему выходу, поэтому выполнение алгоритма будет продолжаться вне зависимости от логического значения условия.

Выделяют два вида развилок — полные и неполные. Для первых указываются действия для случая, когда условие как истинно, так и ложно. Неполные развилки включают действия, относящиеся только к случаю истинности условия.

Общая форма записи полной развилки в форме блок-схемы приведена на рис. 1.4, *а*, неполной — на рис. 1.4, *б*. «Да» и «нет» указывают на ветки выполнения, если условие истинно или ложно соответственно. Иногда вместо слов «да» и «нет» ставят символы «+» и «–». Пример использования развилок в конкретной задаче приведен на рис. 1.2.

На естественном языке полная развилка соответствует фразе: «Если *условие* истинно, то выполняются *действия 1*, иначе выполняются *действия 2*». Неполная развилка соответствует фразе: «Если *условие* истинно, то выполняются *действия*».

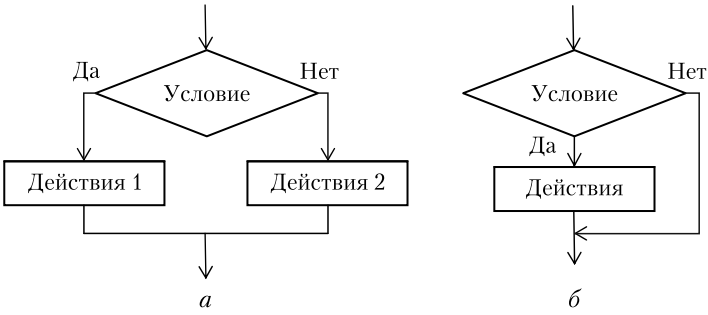


Рис. 1.4. **Развилка**

Цикл. Данная структура предусматривает многократное повторение однотипных действий, называемых **телом цикла**. Каждое выполнение тела цикла называется **итерацией**.

Выделяют три вида циклических структур: цикл с параметром, цикл с предусловием, цикл с постусловием.

Цикл с параметром выполняется заранее известное число раз. Номер текущей итерации цикла хранится в некоторой переменной, называемой **счетчиком цикла**. Имя счетчика часто называют именем цикла. Если, например, счетчик имеет имя `i`, то говорят о «цикле `i`» или «цикле по `i`».

При инициализации цикла задается начальное и конечное значения счетчика, что обеспечивает конечное число итераций. Иногда задается шаг изменения счетчика после каждой итерации.

Общий синтаксис описания счетчика в блок-схемах следующий:

$$имя\_счетчика{:=}\overline{нач\_зн;\ кон\_зн\ [;шаг]},$$

где *нач_зн* и *кон_зн* — начальное и конечное значение счетчика с именем *имя_счетчика*. Квадратные скобки означают, что параметр *шаг* является необязательным (*шаг* не указывается, если он равен 1). Описание счетчика в указанной форме иногда называют *инициализацией цикла*.

В блок-схемах циклические конструкции должны указываться в фигурах, приведенных на рис. 1.5, *а*. В верхней фигуре цикл инициализируется, в нижней указывается имя цикла (имя счетчика). Между ними располагается тело цикла. На рис. 1.5, *б* приведена блок-схема вычисления суммы чисел от 1 до 100 с использованием цикла с параметром.

Цикл с предусловием. Выполнение тела цикла связано с проверкой некоторого условия. На естественном языке выполнение данной БАС можно сформулировать следующим образом: «до тех пор

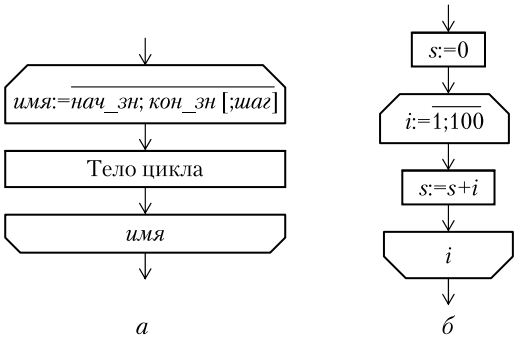


Рис. 1.5. **Цикл с параметром**

пока условие остается истинным, выполнять операторы тела цикла». Другое название этой алгоритмической конструкции — цикл «до».

При практическом использовании этой БАС следует учитывать следующие особенности:

1) число итераций цикла с предусловием неизвестно изначально и определяется в ходе выполнения алгоритма;

2) до выполнения тела цикла условие проверяется как минимум один раз. Если условие изначально имеет значение «ложь», тело цикла не выполнится ни разу;

3) выражение условия может содержать (и обычно содержит) значение некоторой переменной. Изменять ее значение в теле цикла допускается. Более того, без подобного изменения вся конструкция цикла с предусловием теряет смысл. Например, если в качестве условия указано $i > 25$, то при задании начального значения i:=100 и наличия в теле цикла оператора i:=i–1, тело цикла выполнится ровно 76 раз;

4) если условие всегда остается истинным (например, $1 > 0$), то цикл будет выполняться бесконечно.

На рис. 1.6, *а* приведена общая форма записи в соответствии с принятым стандартом; на рис. 1.6, *б* — алгоритм вычисления суммы чисел от 1 до 100 с использованием цикла с предусловием.

Цикл с постусловием. Выполнение тела цикла также связано с проверкой некоторого условия. Тело цикла выполняется, пока условие имеет логическое значение «ложь».

На естественном языке оператор цикла с постусловием можно сформулировать следующим образом: «Выполнять тело цикла, пока условие ложно». Другое название этой алгоритмической конструкции — цикл «пока».

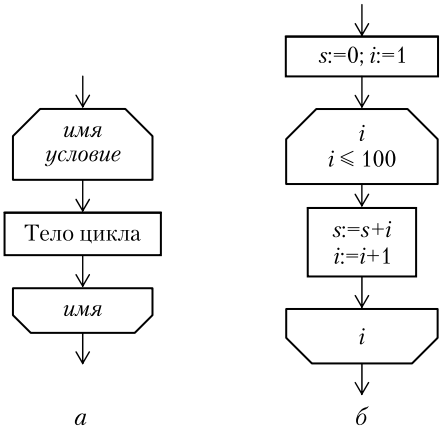


Рис. 1.6. **Цикл с предусловием**

При практическом использовании данного типа цикла следует учитывать следующие особенности:

1) число итераций цикла с постусловием неизвестно изначально и определяется в ходе выполнения алгоритма;

2) тело цикла выполнится как минимум один раз;

3) если условие всегда имеет логическое значение «ложь», например $1 < 0$, цикл будет выполняться бесконечно;

4) выражение условия может содержать (и обычно содержит) значение некоторой переменной. Изменять ее значение в теле цикла допускается.

На рис. 1.7, *а* приведена общая форма записи цикла с постусловием в соответствии с принятым стандартом; на рис. 1.7, *б* — алгоритм вычисления суммы чисел от 1 до 100 с использованием цикла с постусловием.

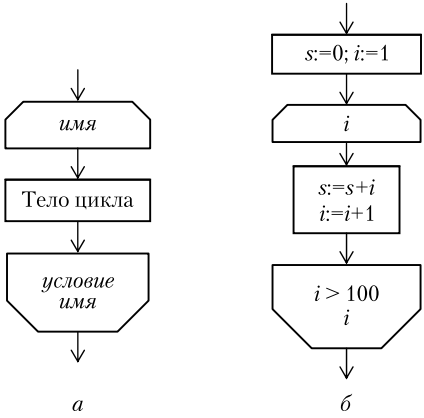


Рис. 1.7. **Цикл с постусловием**

Важно!

Блок-схемы циклов с пред- и постусловием иногда записываются не в соответствии с ГОСТ 19.701—90, хотя и с использованием разрешенных этим стандартом блоков. На рис. 1.8, *а* приведена блок-схема цикла с предусловием, на рис. 1.8, *б* — с постусловием.

Использование базовых алгоритмических структур при разработке алгоритмов важно с практической точки зрения, так как это является основой структурного программирования, которое фактически означает использование только рассмотренных выше БАС. Таким образом, применение в языках программирования инструментария меток, когда произвольный оператор может быть помечен и переход к которому может осуществиться из любого места программы, для структурного программирования недопустимы.

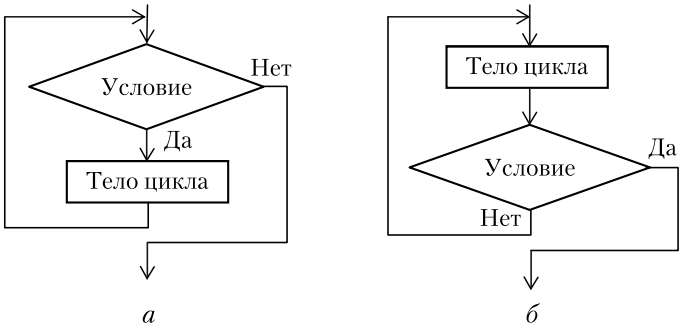


Рис. 1.8. **Альтернативная запись циклов с пред- и постусловием**

Практически все современные языки программирования имеют реализацию БАС в виде отдельных операторов, поэтому в своей основе они являются структурными. Несмотря на появление языков программирования, в основе которых лежат другие концепции (например, объектно-ориентированный подход, используемый в языках Java и C#), рассмотренные структуры в них также реализованы и используются очень часто, так как отражают естественную логику мышления и рассуждений человека, которые закладываются в реализацию хода работы программы.

1.4. Итерационные и рекурсивные алгоритмы

Под **итерацией** в общем случае понимают такой способ организации обработки данных, при котором многократно повторяется некоторая последовательность действий.

Алгоритм, основу которого составляет итерация, называется *итерационным* (*итеративным*). Примером итерационных алгоритмов являются циклические конструкции.

Обобщением понятия итерации является **рекурсия**, под которой понимают такой способ организации обработки данных, при котором алгоритм использует сам себя в качестве вспомогательного алгоритма. Классическими примерами рекурсивных алгоритмов являются вычисление факториала и чисел Фибоначчи.

Факториал целого числа n рекурсивно определяется как

$$n! = \begin{cases} 1, n = 0, \\ n(n-1)!, n > 0. \end{cases} \quad (1.1)$$

Для того чтобы вычислить факториал некоторого числа n, необходимо рассчитать факториал числа $(n - 1)$, для которого, в свою очередь, нужно вычислить факториал числа $(n - 2)$, и т.д.

Обращение алгоритма к самому себе называется *прямым вызовом рекурсии*. Совокупность прямых вызовов иногда называют *рекурсивным спуском*.

После некоторого количества прямых вызовов возникнет необходимость вычислить 1!, значение которого задается определением (1.1). После этого начинается обратный процесс, при котором результат вычисления промежуточного факториала используется для вычисления факториала большего числа. Подобная подстановка значения называется *возвратом рекурсии*. Совокупность рекурсивных возвратов иногда называют *рекурсивным подъемом*.

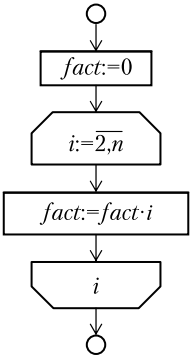


Рис. 1.9. **Итеративное вычисление факториала**

Рекурсивный алгоритм всегда можно преобразовать в итеративный, использующий циклические конструкции. Рекурсивный подход обычно предпочитается итеративному в тех случаях, когда рекурсия более естественно отражает математическую сторону задачи и (или) приводит к алгоритму, который проще для понимания.

Итерационный аналог алгоритма вычисления факториала приведен на рис. 1.9.

Формула (1.1) отражает связь значения факториала некоторого числа n и факториала предыдущего целого числа. Подобные формулы называют рекуррентными.

Формула называется **рекуррентной**, если она выражает зависимость некоторого n-го члена последовательности через предыдущие члены.

Числом Фибоначчи F_m называется число, определяемое рекуррентной формулой

$$F_m = \begin{cases} 0, m = 0, \\ 1, m = 1, \\ F_{m-2} + F_{m-1}, m \geqslant 2. \end{cases}$$

Последовательность первых 10 чисел Фибоначчи выглядит следующим образом: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34. Существует и аналитическое выражение (формула Бине)

$$F_m = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^m - \left(\frac{1-\sqrt{5}}{2}\right)^m}{\sqrt{5}},$$

которое является более громоздким, нежели рекуррентная формула. Нетрудно заметить, что даже при небольших m значения F_m становятся существенными, например, $F_{20} = 6765$, $F_{30} = 832\,040$, $F_{40} = 102\,334\,155$, число F_{300} содержит 63 цифры (в десятичной системе счисления).

При использовании рекуррентных формул для определения факториала и чисел Фибоначчи говорят, что используется *явная рекурсия*, — для вычисления текущего значения используется только исходная рекуррентная формула. При *косвенной* (*неявной*) рекурсии кроме рекуррентной формулы применяются и другие операции.

Примером алгоритма с косвенной рекурсией может быть поиск минимального элемента среди чисел $a_1, a_2, ..., a_n$, в котором минимальный элемент определяется как минимум из двух чисел: последнего элемента и минимального элемента из первых $(n - 1)$ чисел. Нахождение минимального элемента из первых $(n - 1)$ значений осуществляется с помощью аналогичных рассуждений. В конечном итоге поиск минимального элемента сводится к поиску минимального элемента из a_1 и a_2.

Наличие рекурсии при компьютерной реализации алгоритмов требует наличия некоторой памяти — *стека вызова*. В нем хранятся данные о дочерних рекурсивных вызовах, которые используются для возврата управления в родительскую процедуру или функцию.

Процесс разработки рекурсивных алгоритмов состоит из трех этапов: параметризации, выделении базы и декомпозиции.

Параметризация. На этом этапе происходит выделение параметров, описывающих входные данные алгоритма. Например, для вычисления $n!$ это натуральное число n, для вычисления F_m — число m. Для алгоритма поиска минимального элемента во множестве чисел — количество чисел.

Выделение базы предполагает описание тривиальных случаев, для которых результат очевиден и не требует дополнительных операций. Рекурсивный алгоритм в конечном итоге сводится именно к этим случаям. Для рекурсивного вычисления факториала базовым случаем является $0! = 1$, для вычисления чисел Фибоначчи — два базовых случая: $F_0 = 0$ и $F_1 = 1$. Для алгоритма поиска минимального элемента базовым случаем будет нахождение $\min(a_1, a_2)$.

Декомпозиция. На этом этапе рассматривается общий случай выполнения алгоритма, который сводится к выполнению более «простого» случая. Под «простотой» может пониматься снижение значения параметров рекурсивного алгоритма, уменьшение размерности данных и т.п.

При декомпозиции описываются не только связи между общей задачей и подзадачами, но и характер изменения параметров на